設 計 自 動 化 91－16
(1999. 2. 5)

2種類のレジスタファイルを持ったディジタル信号処理向けプロセッサのハードウェア/ソフトウェア協調合成システムとその並列化コンパイラ

中村 剛　　戸川 望　　柳澤 政生　　大附 辰夫

早稲田大学理工学部電子・情報通信学科

〒169-8555 東京都新宿区大久保 3-4-1

E-mail: tsuyoshi@ohtsuki.comm.waseda.ac.jp

あらまし

ディジタル信号処理において高い演算精度を保つためには，演算の途中結果は入力系列のビット幅より大きなビット幅を持つ必要がある．ディジタル信号処理向けプロセッサが2種類のレジスタファイルを持てば，演算精度を保ち，しかも小さいハードウェア面積でディジタル信号処理アプリケーションを実現することができる．本稿では，レジスタビット幅の異なる2種類のレジスタファイルを持ったディジタル信号処理向けプロセッサのハードウェア/ソフトウェア協調合成システムおよびその並列化コンパイラを提案する．本システムはアプリケーションプログラムのC言語記述およびアプリケーションデータを入力とし，プロセッサのハードウェア記述，プロセッサ上で動作するオブジェクトコードおよびソフトウェア環境を出力とする．並列化コンパイラは，与えられたC言語記述からターゲットアーキテクチャで想定される全てのハードウェアユニットを持つプロセッサ上で動作するアセンブリコードを出力する．この際，アプリケーションが持つ並列度を最大限に抽出し，実行時間の最小化を目指す．さらに2つのデータ型から変数を2種類のレジスタファイルに割り当て，演算精度を保つアセンブリコードを生成できる．計算機実験によって本システムを評価した結果を報告する．

キーワード　ハードウェア/ソフトウェア協調合成，コンパイラ，ディジタル信号処理，2種類のレジスタファイル

A Hardware/Software Cosynthesis System for Digital Signal Processors with Two Types of Register Files and Its Compiler

Tsuyoshi NAKAMURA　Nozomu TOGAWA　Masao YANAGISAWA　Tatsuo OHTSUKI

Dept. of Electronics, Information and Communication Engineering

Waseda University

3-4-1 Okubo, Shinjuku, Tokyo 169-8555, Japan

E-mail: tsuyoshi@ohtsuki.comm.waseda.ac.jp

Abstract

In digital signal processing, intermediate results require greater bit width than input data in order to keep high precision for arithmetic operation. If a digital signal processor has two types of register files, digital signal processing applications can keep high precision for arithmetic operation with small amount of processor area. This paper proposes a hardware/software cosynthesis system which synthesizes digital signal processors with two types of register files and its compiler. The input of the system is an application program written in C and application data, and its output is hardware descriptions of a synthesized processor core, an application binary code executed on the processor core and software environment. The proposed compiler generates an assembly code for a processor core with all the available hardware units which can be added to the processor core. It extracts from an input application program those instructions which can be executed concurrently and attempts to minimize its execution time. Moreover it generates an assembly code which keeps required precision for arithmetic operation, since the proposed compiler assigns two types of data to two types of register files. The experimental results show the effectiveness of the system and the compiler.

Key Words *hardware/software cosynthesis, compiler, digital signal processing, two types of register files*

1 まえがき

ディジタル信号処理の基本処理の1つにFIRフィルタがある．n次FIRフィルタは，係数系列$\{a_0, a_1, \cdots, a_{n-1}\}$が与えられたとき，時刻$i$（$i = 0, 1, \cdots$）において印加される入力系列$\{x(0), x(1), \cdots x(i), \cdots\}$に対して，出力系列

$$y(i) = \sum_{0 \le j < n} a_j \times x(i-j) \qquad (1)$$

を得るものである[10]．一般にこのようなn次FIRフィルタをプロセッサによって実現する場合，1個の出力を得るのにn回の乗加算を要する．今，FIRフィルタにおいて係数系列および入力系列がbビットによって与えられたとき，出力系列においてbビットの演算精度を得るには，プロセッサは1回の乗加算結果を$2 \times b$ビット以上のレジスタに保持する必要がある．我々がこれまで提案してきたプロセッサのアーキテクチャモデル[15]あるいはPEAS-I [12]に代表されるプロセッサのアーキテクチャモデルのように，プロセッサが固定されたビット幅を持つ1個のレジスタファイルのみを有する場合，bビットの演算精度を得るには，必要となる最大のレジスタビット幅を考え，$2 \times b$ビット以上のビット幅を持つレジスタファイルを必要とする．ところが，プロセッサが異なるビット幅を持つ2種類以上のレジスタファイルを有する場合，n次FIRフィルタの演算に対して，係数系列，入力系列，出力系列の各々に対してbビットのレジスタを，各乗加算結果に対して$2 \times b$ビットのレジスタを割り当てることが可能である．このようなプロセッサを合成することにより，より小さいレジスタファイルで従来と同等な演算結果を得ることができると考える．

複数のレジスタファイルを持つアーキテクチャは，DSP，メディアプロセッサやRISCなどのうち，マルチメディア処理と呼ばれる高性能演算に対応したプロセッサで多く採用されている．しかしそれらはレジスタを用途別に分けてまとめているのみで，ビット幅は同一のレジスタファイルを用いる例が多い[13]．演算精度を考慮してビット幅の異なる複数のレジスタファイルを持つものに文献[14]がある．

ハードウェア/ソフトウェア協調設計とは，対象とするアプリケーションに応じて，プロセッサあるいはシステムのハードウェア部分とソフトウェア部分とを同時に設計するものである．アプリケーションプログラムから適当なプロセッサ構成を得る協調設計システムに関しては，COACH [1]，PEAS [2],[12]，ASIA [6]の研究例がある．入力となるアプリケーションプログラムから，レジスタビット幅の最適化をはかるハードウェア/ソフトウェア協調設計システムとして文献[7]がある．これは単一のレジスタファイルのビット幅を低消費電力の観点から決定するもので，演算精度とプロセッサコア面積削減に関する考慮はなされていない．

以上のような背景から，本稿では，ビット幅の異なる2種類のレジスタファイルを持ったディジタル信号処理向けハードウェア/ソフトウェア協調合成システム，およびその並列化コンパイラを提案する．システムは文献[15]で提案されたシステムの拡張であり，C言語で書かれたディジタル信号処理アプリケーションプログラム，アプリケーションデータを入力とし，プロセッサコアのハードウェア記述，プロセッサコア上で動作するオブジェクトコード，ソフトウェア環境（コンパイラ，シミュレータ）を出力する．並列化コンパイラは，C言語で書かれたディジタル信号処理アプリケーションプログラム，アプリケーションデータおよびアプリケーション解析結果を入力とし，ターゲットアーキテクチャで想定される全てのハードウェアユニットを持つプロセッサコア上で動作するアセンブリコードを出力する．並列化コンパイラはプロセッサコアが全てのハードウェアユニットを持つと仮定することで，アプリケーションの持つ並列度を最大限に抽出し，実行時間の最小化を目指す．さらに，2種類のレジスタファイルを用いることで，演算精度を保ちながら出力となるプロセッサのハードウェアコストを小さく抑えることができる．

2 2種類のレジスタファイルを持つプロセッサのアーキテクチャモデルと命令セット

ビット幅の異なる2種類のレジスタファイルを持つプロセッサとして，ビット幅がb_1ビットのレジスタファイル1とビット幅がb_2ビットのレジスタファイル2の2種類のレジスタファイルを持つアーキテクチャを考える（図1および図2参照）．このレジスタファイル構成は，独立した2種類のレジスタファイルを持つものであり，以下の2つの利点が挙げられる．

(1) アプリケーションプログラムの要求に応じて，レジスタファイル1内のレジスタ数およびレジスタファイル2内のレジスタ数とを適切に決定する必要があるので，アプリケーションプログラム毎にプロセッサを合成する場合に有効である．

(2) レジスタの割り当て，レジスタファイルのハードウェア構成はレジスタファイル1およびレジスタファイル2に対して独立して考えることができ，比較的容易にコンパイラ，ハードウェア/ソフトウェア分割，ハードウェア生成を実現できる．

以下に，2種類のレジスタファイルを持つプロセッサのアーキテクチャモデルと命令セットを提案する．

2.1 アーキテクチャモデル

プロセッサのターゲットアーキテクチャとして，RISCアーキテクチャを持つ汎用のマイクロプロセッサからディジタル信号処理プロセッサに亘るものを考える．市販のディジタル信号処理プロセッサ[11]をもとに，図1にプロセッサカーネルおよびプロセッサカーネルに付加されるハードウェアユニットの一部を示す．プロセッサカーネルに，いくつかのハードウェアユニットが付加されたものを**プロセッサコア**と呼ぶ．図2は，プロセッサコアのアーキテクチャモデル例である．図2においてレジスタファイル1は，算術演算，論理演算，アドレス演算等の全ての命令セットで使用可能である．レジスタファイル2はアプリケーションの要求により付加されるレジスタファイルであり，そのレジスタビット幅はレジスタファイル1に比較して大きい（$b_1 < b_2$）．乗算命令，乗加算命令等の一部の演算命令によって使用される．(詳細は2.2節を参照)．以下，ハードウェア/ソフトウェア協調合成において，合成の対象となるプロセッサカーネルおよびプロセッサカーネルに付加されるハードウェアユニットを定義する．

プロセッサカーネルとして，(i) RISC型および(ii) DSP型のうちいずれかを取る．RISC型は，文献[5]にならい，IF（命令フェッチ），ID（命令デコード），EXE（命令の実行），MEM（メモリアクセス），WB（書き込み）という5ステージのパイプライン構成をとる．DSP型は，文献[9]–[11]にならい，IF，ID，EXEという3ステージのパイプライン構成をとる．プロセッサカーネルの各候補は，ハーバードアーキテクチャをとり，内部に(c-i) 1個の命令メモリ，(c-ii) 1個のデータメモリ（Xデータメモリ），(c-iii) レジスタファイル1，(c-iv) バレルシフタ，ALU（算術論理演算ユニット）を持つ．(c-i), (c-ii)においてデータバス幅を変化することができる．この構成により，表1に示す必須命令を実行可能となる．なお，命令メモリ，Xデータメモリ（およびYデータメモリ）のアドレスバス幅は16ビットに固定される．さらに，プロセッサカーネルの各候補は，複数の命令を同時に実行することができる．同時に実行可能な命令数は，予め与えられ固定とする．

プロセッサカーネルがRISC型をとったとき，パイプラインステージ数が多く高速動作可能である．反面，ハードウェアルー

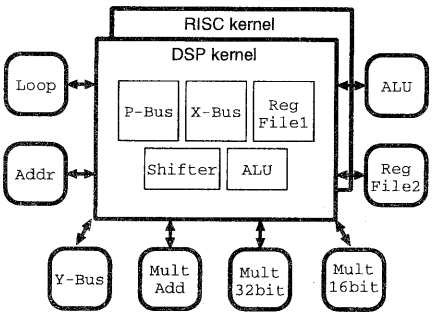

図 1: プロセッサカーネルおよび付加されるハードウェアユニット.

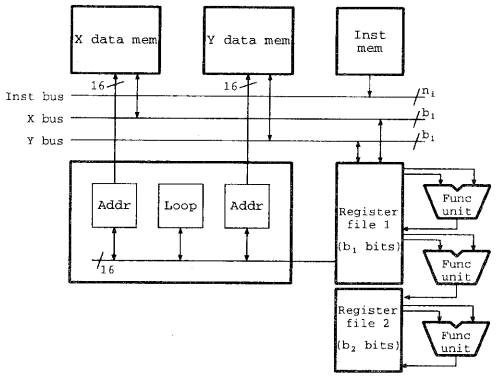

図 2: 2種類のレジスタファイルを持つプロセッサのアーキテクチャモデル.

プ，アドレッシングユニットの制御に適さない．プロセッサカーネルがDSP型をとったとき，パイプラインステージ数が少なくRISC型に比較して動作が低速である．反面，ディジタル信号処理に特有のハードウェアループ，アドレッシングユニットを有することができる．このようなプロセッサカーネルを用意することにより，RISCアーキテクチャの汎用マイクロプロセッサから汎用のディジタル信号処理プロセッサを合成することができる．

Yデータメモリおよびこれに付随するYバスおよびアドレッシングユニットの有無を選択できる．Yデータメモリを使用する場合，Xデータメモリおよび Yデータメモリを用いることで，データメモリから（に）並列に2種類の値をロード（ストア）することができる．

演算ユニットの種類および個数を変化させる．演算ユニットとして，シフタ，ALU，乗算器，除算器，乗加算器を考える．ALUはシフト命令および乗算，除算，乗加算以外の演算を実行できるものとする．乗加算器は乗算および加算を単独で実行することができる．

アドレッシングユニットとして，(i) no operation, (ii) post increment, (iii) post decrement, (iv) index add, (v) modulo add, (vi) bit reverse のアドレス演算を実現できるものを考える[1]．

レジスタファイル2の有無および**レジスタ数**を変化させる．レジスタとして，レジスタファイル1および2内の汎用レジスタ，X，Yデータメモリ用のアドレッシングユニット内のアドレスレジスタ，インデックスレジスタおよびモジュロレジスタ，ハードウェアループ内のネストレジスタの各々を考える．アドレッシングユニット内のアドレスレジスタおよびインデックスレジスタ数は同一であり，しかもX，Yデータメモリで同数とする．モジュロレジスタは，X，Yデータメモリ用に高々1個ずつ用意される．

データメモリのデータバス幅を変化させる．これは，入力となるアプリケーションのC言語内で設計者が設定する．Xデータメモリのデータバスと Yデータメモリのデータバスが存在する場合，これらは同一のビット幅とする．レジスタファイル1および2内のレジスタのビット幅はそれぞれアプリケーションプログラム内で定義する．演算ユニットのビット幅は，ハードウェア/ソフトウェア分割時に決定される．

命令メモリのデータバス幅は，インストラクションセットから決定される．

各ハードウェアユニットは**ハードウェアコスト**を持つ．ハードウェアコストとして面積を考える．より多くのハードウェアユニットを持つプロセッサコアはハードウェアコストが増大しより多くのチップ面積を必要とする．

[1]各アドレス演算の処理の詳細は，例えば文献 [9], [10] を参照されたい.

2.2 命令セット

2種類のレジスタファイルを持つプロセッサの命令セットを提案する．合成されるプロセッサは命令セットとして基本命令および複合命令を持つ．基本命令は汎用のディジタル信号処理プロセッサの命令セット [9]–[11] をもとに，以下の方針を取る．複合命令は基本命令を複数個並列実行する命令である．

(1) レジスタファイル1内のレジスタは，従来の命令セット [15] の全てを引き継ぐ．レジスタファイル2内のレジスタは，乗算命令，乗加算命令等の一部の演算命令によって使用される．

(2) レジスタファイル1内のレジスタが期待されるオペランドにレジスタファイル2内のレジスタが指定された場合，適当な b_1 ビットを切り出す．同様に，レジスタファイル2内のレジスタが期待されるオペランドにレジスタファイル1内のレジスタが指定された場合，b_2 ビットに符号拡張する．

このような方針を取ることによって，レジスタファイル1のみから構成されるプロセッサの基本アーキテクチャに，部分的にレジスタファイル2を付加したプロセッサアーキテクチャの構成を得られる．

算術命令のLONG型対応命令において，MULL，MULILおよびMACLは，ソースとしてレジスタファイル1内のレジスタを，デスティネーションとしてレジスタファイル2内のレジスタをとる．例えば`MACL R1, R2, R3L`は，レジスタファイル1内のレジスタ`R1`, `R2`を乗算関数`mul()`（詳細は3.2節を参照）により，必要な演算精度を保ったまま乗算し，その結果とレジスタファイル2内のレジスタ`R3L`を加算する命令である．その他のLONG型対応命令は，ソース，デスティネーション共にレジスタファイル2内のレジスタを用いる．`ADDL R1L,R2L, R3L`は，レジスタファイル2内のレジスタ`R1L`, `R2L`の加算をレジスタファイル2内のレジスタ`R3L`に格納する命令である．ロード/ストア命令のLONG型対応命令は，レジスタファイル2内のレジスタを用いて実行される．`MVL R1L, R2L`は，レジスタファイル2内のレジスタ同士の移動命令であり，`IMML R1L, imm`はレジスタファイル2内のレジスタ`R1L`に即値`imm`を格納する命令である．

表1に基本命令の一覧を示す．

3 2種類のレジスタファイルを持つプロセッサの合成

C言語で書かれたアプリケーションプログラムおよびアプリケーションデータを入力したとき，2章で導入した2種類のレジスタファイルを持つディジタル信号処理向けプロセッサの自動合成を考える．

表 1: 基本命令（下線のある命令は必須命令）.

基本命令 1 （算術命令）	ADD, SUB, SRA, SRL, SLL, AND, OR, XOR, MUL, DIV, SLT, SEQ, SNE, COM2, MAC, INC, DEC, ADDI, SUBI, SRAI, SRLI, SLLI, ANDI, ORI, XORI, MULI, DIVI
≪LONG 型対応命令 ≫	ADDL, SUBL, MULL, MACL, SRAL, SRLL, SLLL, INCL, DECL, ADDIL, SUBIL, SRAIL, SRLIL, SLLIL, MULIL,
基本命令 2 （ロード/ストア命令）	LDX, LDY, STX, STY, LDRX, LDRY, STRX, STRY, LDXI, LDYI, STXI, STYI, LDIX, LDIY, STIX, STIY, MV, IMM
≪LONG 型対応命令 ≫	MVL IMML
基本命令 3 ジャンプ命令他	BEQ, BNE, BZ, BNZ, JP, LOOP, RPT, CALL, RET, NOP, HLT
基本命令 4 並列ロード/ストア命令	LDPX, STPX

まず，ビット幅の異なる 2 種類のレジスタファイルに対応して，C 言語におけるビット幅の異なる 2 種類のデータ型 INT 型および LONG 型[2] を考え，INT 型の変数をレジスタファイル 1 内のレジスタに，LONG 型の変数をレジスタファイル 2 内のレジスタに対応付ける．INT 型のビット幅 b_1 および LONG 型のビット幅 b_2 をアプリケーションプログラムの中で明示的に与え，レジスタファイル 1 のレジスタビット幅を b_1 ビット，レジスタファイル 2 のレジスタビット幅を b_2 ビットとする．

一般的なディジタル信号処理ではアプリケーションプログラムに応じてレジスタファイル 1 およびレジスタファイル 2 に適当な固定小数点位置が要求される．即ち，INT 型および LONG 型間の型変換および乗算結果は，アプリケーションプログラムに応じて変化させる必要がある．

ここでは，ビット幅の異なる 2 種類の型に起因する問題を解決するため，C 言語に次のような型変換関数および乗算関数を提案する．最後に FIR フィルタの C 言語記述例を示す．

3.1 型変換関数

INT 型および LONG 型間の型変換に対して型変換関数 (INT_to_LONG(), LONG_to_INT()) を用いる．アプリケーションプログラム内で，LONG 型の変数のうち INT 型に対応するビットを明示的に指定する．LONG 型から INT 型に変換する場合，LONG 型のうち指定されたビットを INT 型として切り出す．INT 型から LONG 型に変換する場合，LONG 型のうち指定されたビットに INT 型を対応させ，その他の部分は符号拡張および 0 拡張する（図 3）．アプリケーションプログラムにおいて，非明示的な型変換は許されず，型変換関数によってのみ型変換が実行される．以下に，型変換関数を定義する．いま，x をレジスタファイル 1 内のレジスタ，yl をレジスタファイル 2 内のレジスタ，INT 型のビット幅を b_1，LONG 型のビット幅を b_2 とする．LONG 型の b_{low} ビットから b_{high} ビットまでを INT 型に対応させるとき，**型変換関数 INT_to_LONG(x)** とは，b_1 ビット変数 x を b_{low} ビット左シフトし，$0 - b_{low} - 1$ ビットは 0 拡張，$b_{high} - b_2 - 1$ ビットを x の第 $b_1 - 1$ ビットで符号拡張する関数とする．一方，**型変換関数 LONG_to_INT(yl)** とは，b_2 ビット変数 yl を b_{low} ビット右シフトし，b_1 ビットの値をとりだす関数である．定数 b_1，b_2，b_{low} および b_{high} は，アプリケーションプログラム内に記述される．

INT 型および LONG 型の型変換に対応し，命令セットにおいて b_2 ビットのレジスタが要求されるとき，b_1 ビットのレジスタは関数INT_to_LONG() の仕様に従ってハードウェアによって符号拡張される．同様に，命令セットにおいて b_1 ビットのレジスタが要求されるとき，b_2 ビットのレジスタは関数LONG_to_INT() の仕様に従ってハードウェアにより b_1 ビットが切り出される．

以上の処理により，アプリケーションプログラムをアセンブリコードに変換する際は，INT 型および LONG 型にそれぞれレジスタファイル 1 およびレジスタファイル 2 のレジスタを割り当てればよい．

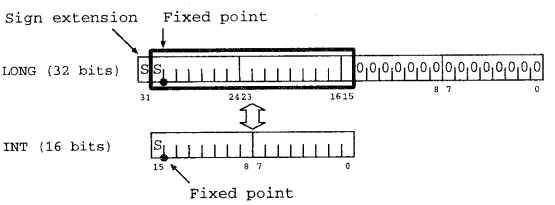

図 3: INT 型と LONG 型の変換.

3.2 乗算関数

乗算として，(1) INT 型同士の乗算で結果が INT 型となるもの，(2) INT 型同士の乗算で結果が LONG 型となるもの，(3) LONG 型同士の乗算で結果が LONG 型となるものの 3 種類を考えることができる．(1),(3) は，C 言語の言語仕様による乗算であり，(2) は，演算精度を考え導入された乗算である．入力となる C 言語では，(3) による乗算は考えず，(1),(2) のみを乗算として扱う．(1),(2) を区別するため，(1) は 2 項演算子「*」，(2) は関数mul() によって実現する．2 項演算子「*」は小数点位置が最下位ビットにあることを想定した INT 型同士の通常の C 言語の乗算である．関数mul() を次のように定義する．まず，INT 型のビット幅を b_1，LONG 型のビット幅を b_2 とし，INT 型が p_{int} ビットと $p_{int} - 1$ ビットの間，LONG 型が p_{long} ビットと $p_{long} - 1$ ビットとの間に固定小数点が存在するとみなす．x，y をレジスタファイル 1 内のレジスタ，zl をレジスタファイル 2 内のレジスタとするとき，**乗算関数 mul(x,y)** は x，y の b_1 ビット以降を 0 拡張して得られる b_2 ビットの lx と ly の乗算結果から，p_{int} と p_{long} の大小関係より固定小数点を補正した値を得る関数である．命令セットにおいて，MUL，MULI あるいはMAC は，INT 型の 2 項演算子「*」に対応した動作を実行する．MULL，MULIL，あるいはMACL は，上述の関数mul() に対応した動作を実行する．

3.3 FIR フィルタの C 言語記述例

本章で議論した，型変換関数および乗算関数を用いて n 次 FIR フィルタの C 言語記述例を以下に示す．10 行目において変数tmp が LONG 型として宣言される．変数tmp は合成されたプロセッサではレジスタファイル 2 のレジスタに割り当てられる．14 行目においてmul(a[j], x[i - j]) は，上述の関数mul() の意味においてa[j] とx[i - j] との乗算を実行し，結果を LONG 型として返す．その値は，LONG 型の変数tmp に LONG 型の加算によって加えられる（tmp += ...）．15 行目において，変数tmp のうち，適切な b_1 ビットがy[i] に転送される．

```
1  INT     a[N] /* = { coefficients } */;
2  INT     x[M+N-1]; /* input data */
3  /* y[i] = \sum_{0 \geq j < N} a[j] * x[i-j] */
4  INT     y[M+N-1];
5  int     main(void)
6  {
7      INT     loop;
8      INT     i;
```

[2] C 言語仕様上の int 型および long 型との区別を明確にするために INT 型および LONG 型と記述している．

```
 9      INT     j;
10      LONG    tmp;
11      for (i = N - 1; i < M + N - 1; i++) {
12          tmp = 0;
13          for (j = 0; j < N; j++)
14              tmp += mul(a[j], x[i - j]);
15          y[i] = LONG_to_INT(tmp);
16      }
17      return 0;
18  }
```

このような n 次 FIR フィルタの C 言語記述の 14 行目および 15 行目は，次のようなアセンブリ言語記述にコンパイルされる．14 行目は，MACL 命令，LDIX 命令，LDIY 命令の複合命令として実現される．LDIX 命令は，X メモリ用アドレスレジスタDPX1 を用いたロード命令で，X メモリのDPX1 番地の値をレジスタファイル 1 内のレジスタR1 に読み込み，その後DPX1+1 を実行する．LDIY 命令は，Y メモリ用のアドレスレジスタDPY1 を用いて，Y メモリのDPY1 番地の値をレジスタファイル 1 内のレジスタR2 に読み込み，その後DPY1-1 を実行する命令である．15 行目は，STIX 命令のみによって実現される．型変換関数LONG_to_INT() は，STIX 命令が暗黙のうちに実行する．STIX 命令は，型変換関数によりレジスタファイル 2 内のレジスタR1L の b_1 ビットを，X メモリのDPX2 番地に格納する命令である．

```
/*** (line 14) ***/
// tmp += mul(a[j], x[i - j]);
 MACL R1, R2, R1L || LDIX  1, R1, DPX1 || LDIY  2, R2, DPY1

/*** (line15) ***/
// y[i] = LONG_to_INT(tmp);
 STIX  1, R1L, DPX2
```

4 ハードウェア/ソフトウェア協調合成システム

ディジタル信号処理向けプロセッサを対象としたハードウェア/ソフトウェア協調合成システムは，C 言語で書かれたディジタル信号処理アプリケーションプログラム，アプリケーションデータを入力とし，プロセッサコアのハードウェア記述，プロセッサコア上で動作するオブジェクトコード，ソフトウェア環境（コンパイラ，シミュレータ）を出力する．アプリケーションプログラムを実行する時間を制約とし（時間制約と呼ぶ），面積最小のプロセッサコア合成を目的とする．プロセッサコアの面積は，図 1 で表されたプロセッサカーネルおよびプロセッサカーネルに付加される各ハードウェアユニットのうち，プロセッサコアに使用されるものの面積の和によって与えられる．アプリケーションプログラムの実行時間は，アプリケーションプログラムを実行するのに必要なクロックサイクル数とクロック周期との積によって与えられる．

図 4 に提案するシステムを示す．各要素技術のうち，核となるコンパイラ，ハードウェア/ソフトウェア分割およびハードウェア生成を，2 種類のレジスタファイルを持つプロセッサを自動合成するために次のように拡張する．

コンパイラ: アプリケーションの実行に必要とされるすべてのハードウェアユニットを持つと仮定したプロセッサコア上で，アプリケーションの持つ最大限の並列性を抽出したアセンブリコードを生成する．複数のレジスタファイルを持つプロセッサに対しては，2 つのデータ型の扱い，2 種類のレジスタファイルを考慮したレジスタ割り当て，LONG 型対応命令セットに関する考慮が必要となる（詳細は 5 章を参照）．

ハードウェア/ソフトウェア分割: コンパイラで得られたアセンブリコードは，アプリケーションプログラムの実行時間は短いが，プロセッサコアの面積が大きくなる．ハードウェア/ソフトウェア分割では，ハードウェアによる実現部の一部を徐々にソフトウェアによって代替することで，プロセッサコアの面積削減を

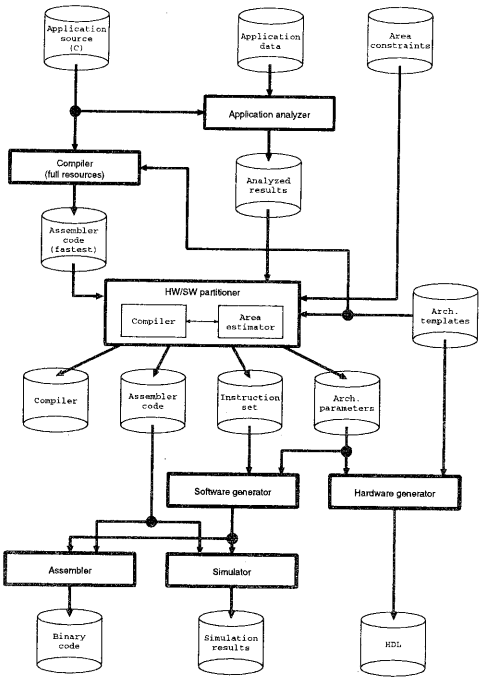

図 4: ディジタル信号処理向けプロセッサのハードウェア/ソフトウェア協調合成システム．

図る．この操作は時間制約を満たす限り繰り返され，最終的にアプリケーションの実行時間が時間制約を満たし小面積のプロセッサコアを合成することが可能となる．ハードウェア/ソフトウェア分割では，新たなハードウェアユニットとしてレジスタファイル 2 を導入する．

ハードウェア生成 ハードウェア生成は，ハードウェア/ソフトウェア分割の結果から VHDL 記述を自動生成する．2 種類のレジスタファイルを持つプロセッサに対応して，生成されるプロセッサコアのパイプラインユニットは全て LONG 型のビット幅を持つ．INT 型の変数は，ビット拡張されてパイプラインレジスタに格納される．INT 型が b_1 ビット，LONG 型が b_2 ビットのとき，INT 型の変数が INT 型用の演算器の入力となる際は，ビット切り出し器により b_1 ビットを切り出されて演算され，その後ビット拡張器により b_2 ビットに戻される．つまり，ビット拡張やビット切り出しは各演算器およびレジスタの前後で実行される．この手法により，アプリケーションの実行に際して，フェッチした命令のオペランドが INT 型か LONG 型かを区別する必要がなくなる．さらに，2 種類のレジスタファイルを持つプロセッサへの対応を，ビット切り出し器およびビット拡張器を持った演算器と 2 種類のレジスタファイルの導入と捉えることができ，従来のハードウェア見積り式 [4] を適用できる．

このようにして最終的に，プロセッサコアのハードウェア記述，プロセッサコア上で動作するアプリケーションプログラムのオブジェクトコードおよびソフトウェアを生成する（ハードウェア生成，ソフトウェア生成）．

5 並列化コンパイラ

本ハードウェア/ソフトウェア協調合成システムにおける並列化コンパイラは，C 言語で記述されたディジタル信号処理アプリケーションプログラムから，ハードウェアユニットや並列性を抽

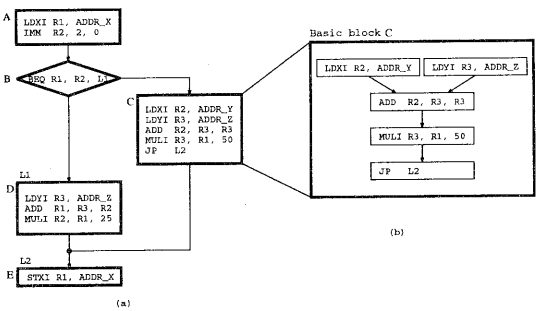

図 5: コントロールフローグラフ (a) とデータフローグラフ (b)

出し，プロセッサアーキテクチャのハードウェアユニットを最大限利用した，最速のアセンブリコードを生成する．提案する並列化コンパイラの特徴は，(1) 2つのデータ型を2種類のレジスタファイルに割り当て，(2) コンパイルテンプレートにより，ディジタル信号処理プロセッサに特有のハードウェアユニットを抽出することである．並列化コンパイラの出力は，システムにおいて次段のハードウェア/ソフトウェア分割の入力データとなる．

5.1 問題定義

コールグラフ $G_c = (V_c, E_c)$ とは，アプリケーションプログラムに含まれる関数間の呼び出し関係を表したグラフである．G_c の各節点 $v \in V_c$ は個々の関数に対応する．関数 v_1 が関数 v_2 を呼ぶとき，G_c は有向枝 (v_1, v_2) を持つ．簡単のため，アプリケーションプログラムは再帰呼び出しを含まないとする．即ち，G_c としてサイクルのない有向グラフを仮定する．コールグラフの各節点には，コントロールフローグラフ（以下，CFG）が対応する．コントロールフローグラフ $G_{cf} = (V_{cf}, E_{cf})$ とは，各関数内部における制御の流れを表すグラフである．最初および最後を除いて制御の分岐，結合が存在しない，一連の命令列を**基本ブロック**と呼ぶ．G_{cf} の各節点 $v \in V_{cf}$ は，基本ブロックに対応する．基本ブロック v_1 から基本ブロック v_2 への制御の流れが存在するとき，G_{cf} は有向枝 (v_1, v_2) を持つ．コントロールフローグラフの各節点には，データフローグラフ（以下，DFG）が対応する．データフローグラフ $G_{df} = (V_{df}, E_{df})$ とは，各基本ブロックのデータの流れを表すグラフである．G_{df} の各節点 $v \in V_{df}$ は，1つの基本命令に対応する．基本命令 v_2 が基本命令 v_1 にデータ依存があるとき，G_{df} は有向枝 (v_1, v_2) を持つ．アプリケーションプログラム中の全ての関数，基本ブロックの集合を各々 $\mathcal{F}_{app}$ および $\mathcal{B}_{app}$ とする．図5に CFG と DFG の例を示す．同図 (a) は A, B, C, D, E の5個の基本ブロックから構成されるコントロールフローグラフを表し，同図 (b) は基本ブロック C に対応するデータフローグラフを表す．

アプリケーションプログラム解析系で測定された各基本ブロック $B \in \mathcal{B}_{app}$ の実行回数を N^B_{exe} とする．合成されたプロセッサが基本ブロック B の実行に要するクロックサイクル数を N^B_{cycle} とすれば，アプリケーションプログラムの実行に必要な総クロックサイクル数 N_{cycle} は，$N_{cycle} = \sum_{B \in \mathcal{B}_{app}} N^B_{exe} \cdot N^B_{cycle}$ となる．

以上の準備のもと，並列化コンパイラ問題は次のように定義される．

定義 1 並列化コンパイラ問題とは，入力として C 言語で記述されたアプリケーションプログラム，プロセッサアーキテクチャと命令セット，および基本ブロックの実行回数が与えられたとき，アセンブリコード (コールグラフ，CFG，DFG) を出力することである．プロセッサアーキテクチャと命令セットを制約とし，アプリケーションプログラムの実行に要する総クロックサイクルの最小化を目的とする．

5.2 アルゴリズム

並列化コンパイラは，コールグラフを構築しながら CFG を構築し，CFG を構築しながら DFG を構築する．コールグラフを構築することは，アプリケーションプログラムから関数を抽出することにあたる．CFG を構築することは，関数を基本ブロックに分割することにあたる．DFG を構築し，スケジューリングをすることにより，基本ブロック毎にアセンブリコードを生成する．ターゲットアーキテクチャで想定される全てのハードウェアユニットが利用可能であると仮定することにより，アプリケーションプログラムの持つ並列性を抽出する．

並列化コンパイラのアルゴリズムを以下に示す．

入力: プリプロセス済みのアプリケーション C 言語記述，アプリケーションデータ，アプリケーション解析結果

出力: ターゲットアーキテクチャで想定される全てのハードウェアユニットを持つプロセッサ上で動作するアセンブリコード

Step 1. アプリケーションの C 言語記述を逐次的に読み込みながら以下の処理を実行する．

- **Step 1.1** 関数宣言あるいは関数呼出からコールグラフを構築する．
- **Step 1.2** グローバル変数宣言から，変数の管理のために，変数名と仮に割り当てたメモリアドレスの管理表を用意する．
- **Step 1.3** if 文，while 文，for 文から CFG を構築する．
- **Step 1.4** CFG 内の C 言語記述を逆ポーランド記法と等価な解析木に変換する．ローカル変数宣言の場合は，変数の管理のために，変数名とレジスタ番号の管理表を用意する．
- **Step 1.5** CFG 毎に解析木と変数の管理表から DFG を構築する．このとき，コンパイルテンプレートを用いて DSP 特有のハードウェアユニットを抽出し，DFG に変換する．

Step 2. DFG 内の各節点をスケジューリングする．

Step 3. 実メモリアドレスを決定する．

Step 4. 実行クロックサイクル数を算出する．

上記のアルゴリズムにおいて Step 1.1，1.3，および Step 4 は簡単に処理できる．よって，Step 1.2 (5.2.1 節)，Step 1.4 (5.2.2 節，5.2.1 節)，Step 1.5 (5.2.3 節)，Step 2. (5.2.4 節)，Step 3 (5.2.5 節) について説明する．

5.2.1 変数の管理 (Step 1.2, 1.4)

用語を定義する．**変数レジスタ**は，汎用レジスタのうち，関数内のローカル変数の値を保持しているものである．**自由レジスタ**は，汎用レジスタのうち，変数レジスタ以外のレジスタである．自由レジスタは，演算の途中結果やメモリに格納されている変数の値を読み込み演算処理する場合に用いられる．変数の型には INT 型および LONG 型がある．INT 型をレジスタファイル1内のレジスタに，LONG 型をレジスタファイル2内のレジスタに割り当てる．グローバル変数はメモリに格納する．実際のメモリアドレスが決定されるのはハードウェア/ソフトウェア分割後であるので，コンパイル中は表中の番号を仮のメモリアドレスとして使用する．X，Y メモリの割り当てはスケジューリング後に決定される．ローカル変数は変数レジスタに格納する．

5.2.2 解析木の構築 (Step 1.4)

逆ポーランド記法と等価な解析木を C 言語記述から逐次的に構築する．

5.2.3 DFG の構築（Step 1.5）

DFG は C 言語記述から構築した解析木と変数の管理表から CFG 毎に構築される．提案する並列化コンパイラは，通常のコンパイラによるコンパイル処理に加え，2 種類のレジスタファイルに対応する処理およびコンパイルテンプレートを利用した DFG 構築をする．

まず，2 種類のレジスタファイルに対応した処理を考える．2 種類のレジスタには，型変換関数および乗算関数`mul()` を導入することで対処する．型変換関数は，ハードウェアが自動的に実行するために，コンパイラは型変換関数を意味する DFG を構築しない．`mul()` 関数は MULL，MULIL あるいは MACL 命令に変換する．`mul()` 関数の結果を保存するレジスタとして LONG 型レジスタを割り当てるものとする．

DSP 特有のハードウェアユニットと対応させる一連の C 言語記述をコンパイルテンプレートと呼ぶ．DSP 特有のハードウェアユニットであるハードウェアループおよびアドレッシングユニットと C 言語記述の対応を考える．ハードウェアループは，以下のコンパイルテンプレートにより for 文と対応をとる．

```
for  (  i = const  ;  i < const  ;
        i++  or  i += const  or
        i--  or  i -= const)  state
```

アドレッシングユニット内の機能は，ハードウェアループによって実現される for 文の実行文中で以下の配列と対応をとる．i は現在のループ制御変数，j はループ内では不変の変数，N は定数とする．

```
x[i],  x[i+N],  x[i-N],
x[N*j+i],  x[N*i+j],
x[j][i],  x[i][j]
```

なお，レジスタファイル 2 内のレジスタ数はコンパイラで決定されハードウェア/ソフトウェア分割時に削減されないため，コンパイラが削減を考慮しなければならない．レジスタ数の削減は文献 [8] による．

5.2.4 スケジューリング（Step 2）

全関数の全基本ブロックについて DFG を構築後，各 DFG に対して，リストスケジューリング [3] により DFG 内の各節点をスケジューリングする．リソース制約は，同時に実行される命令数が与えられた並列度以下であること，同時にアクセスされるグローバル変数が 2 つまでであることである．

5.2.5 実メモリアドレス（Step 3）

X，Y メモリの具体的なメモリアドレスを実メモリアドレスと呼ぶ．Step 1.2 で管理表に登録されたメモリアドレスは番号付けのみで，実メモリアドレスは指定されていない．実メモリアドレスは，スケジューリング後に同時にアクセスされる変数の組のうち，アクセス回数の多い組から順に X，Y メモリへ割り当てられる．その結果，X，Y メモリ同時にアクセスできる回数が増え，同時に実行される命令数が増加する．

6 計算機実験結果

本システムは，Sun Ultra 1 上に C 言語を用いて実装されている．本システムを 2 次元 DCT（Discrete Cosine Transform, 8×8 画素），1024 次 FIR フィルタ（10000 要素）および行列積（Matrix，2 個の 100 × 100 行列の乗算）の 3 種類のアプリケーションプログラムに適用した．提案したコンパイラには最大並列度として 1，2，∞ の 3 通りを与え，レジスタファイル 1 のみを持つプロセッサのコンパイル結果と 2 種類のレジスタファイルを持つプロセッサのコンパイル結果を比較する．また最大並列度 1，2 の場合，レジスタファイル 1 のみを持つプロセッサの合成結果と 2 種類のレジスタファイルを持つプロセッサの合成結果からプロセッサのハードウェアコストを比較する．表 2 に本システムのハードウェアユニットライブラリの一部を示す．これらは，VDEC ライブラリ（0.35μm テクノロジー）を用いて合成した．

表 2: ハードウェアユニットライブラリ．

Hardware unit		Bit width	Clock cycles	Area [μm^2]	Delay [ns]
Kernel	DSP-type	32	—	474111*	—
	RISC-type	32	—	605737*	—
Functional unit	ALU	32	1	151194	3.25
	Shifter	32	1	96838	1.35
	ADD	32	1	41963	2.49
	Multiplier	16	1	356948	5.71
	MAC	32	1	463716	7.28
	ALU	16	1	78915	2.82
	Shifter	16	1	38859	1.02
	ADD	16	1	25259	1.44
Addressing unit	(i)	16	1	16346	—
	(i)–(iii)	16	1	20154	—
	(i)–(iv)	16	1	30535	—
Hardware loop		—	—	156344	—
Bit extraction		32→16	—	48	—
Register	General	16 * 8	—	126558	1.14
	General	32 * 8	—	262730	1.65
	DP, DN and DMX	16	—	102107	1.15
	Loop	16	—	76809	—

Addressing unit (i): no operation, (ii): post increment, (iii): post decrement, and (iv): index add

* $n = 2$, $m = 1$ and there are an ALU, a shifter, and two registers in the processor core

コンパイル結果比較を表 3 および表 4，プロセッサのハードウェアコスト比較を表 5 に示す．表 3 および表 4 を比較すると，2 つのコンパイル結果は並列実行命令数や実行サイクル数の点でほぼ変化は無い．しかし，2 種類のレジスタファイルを持つプロセッサのコンパイル結果では，高い演算精度を必要とする変数をレジスタファイル 2 のレジスタに割り当て，それ以外はレジスタファイル 1 のレジスタを用いるコードを生成している．表 5 から，2 種類のレジスタファイルを持つプロセッサは，レジスタファイル 1 のみのプロセッサと比較して，最大で 16.0%，平均して 11.4%ハードウェアコストの小さいプロセッサを生成している．これは，2 種類のレジスタファイルを持つプロセッサが，1 つのレジスタファイルのみを持つプロセッサのレジスタビット幅よりも小さいビット幅を持つレジスタファイルと，そのためのハードウェアユニットを用いることによりハードウェアコストを削減したためである．

以上，提案したシステムは，ターゲットアーキテクチャが 2 種類のレジスタファイルを持つことにより，演算精度と性能を保ちながらハードウェアコストの小さいプロセッサを合成できることが示された．

7 むすび

ビット幅の異なる 2 種類のレジスタファイルを持ったディジタル信号処理向けプロセッサの合成手法およびそのコンパイラを提案した．現在，本システムは時間制約のみをとっているが，今後，消費電力の制約，ハードウェアユニットの数あるいは種類の制約等を導入していく．並列化コンパイラはターゲットアーキテクチャ，命令セットの制約に加え，消費電力を制約に導入していく．

謝辞

本研究に関し御協力いただいた本学片岡義治氏，桜井崇志氏及び吉澤大氏に感謝致します．本研究の一部は，文部省科学研究費補助金（基盤研究 C（2），課題番号 10650345 および奨励研究（A），課題番号 10750303）の援助を受けた．

表 3: アプリケーションプログラムのコンパイル結果（2Reg）

アプリケーション	DCT			FIR			Matrix		
並列度	1	2	∞	1	2	∞	1	2	∞
最大同時発行基本命令数	1	2	4	1	2	4	1	2	4
最大同時発行 ALU 命令数	1	1	2	1	1	1	1	1	2
最大同時発行乗算命令数	1	1	2	0	0	0	1	1	2
最大同時発行乗加算命令数	1	1	1	1	1	1	1	1	1
最大同時発行シフト命令数	1	1	1	1	1	1	0	0	0
ハードウェアループネスト数	3	3	3	2	2	2	3	3	3
Y メモリ	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes
アドレスレジスタ数	12	12	12	4	4	4	3	3	3
インデックスレジスタ数	0	0	0	0	0	0	1	1	1
INT 型レジスタ数 (自由レジスタ数)	5(3)	5(3)	5(3)	4(3)	4(3)	4(3)	5(3)	5(3)	5(3)
LONG 型レジスタ数	1	1	1	1	1	1	1	1	1
実行命令数	2398	2950	4293	10290003	20550005	30800005	1060503	2100704	3110704

表 4: アプリケーションプログラムのコンパイル結果（1Reg）

アプリケーション	DCT			FIR			Matrix		
並列度	1	2	∞	1	2	∞	1	2	∞
最大同時発行基本命令数	1	2	4	1	2	4	1	2	4
最大同時発行 ALU 命令数	1	1	2	1	1	1	1	1	2
最大同時発行乗算命令数	1	1	2	0	0	0	1	1	2
最大同時発行乗加算命令数	1	1	1	1	1	1	1	1	1
最大同時発行シフト命令数	1	1	1	1	1	1	0	0	0
ハードウェアループネスト数	3	3	3	2	2	2	3	3	3
Y メモリ	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes
アドレスレジスタ数	12	12	12	4	4	4	3	3	3
インデックスレジスタ数	0	0	0	0	0	0	1	1	1
レジスタ数 (自由レジスタ数)	6(3)	6(3)	6(3)	5(3)	5(3)	5(3)	6(3)	6(3)	6(3)
実行命令数	2464	3142	4357	10029003	20550005	30800005	1060503	2100704	3110704

表 5: 実験結果.

		並列度	Area [μm^2]	T [ns]	Hardware configuration			
					Kernel	#FUs	#Regs	Addr unit
DCT	2Regs	1	1950730	10	DSP	(1,1,0,1)	(5,1,14,3)	(i)+(ii)+(iii)
		2	2115451	11	DSP	(1,1,0,1)	(5,1,14,3)	(i)+(ii)+(iii)
	1Regs	1	2111802	11	DSP	(1,1,0,1)	(6,0,14,3)	(i)+(ii)+(iii)
		2	2322956	11	DSP	(1,1,0,1)	(6,0,14,3)	(i)+(ii)+(iii)
FIR	2Regs	1	1536568	11	DSP	(1,1,0,1)	(4,1,4,2)	—
		2	1760271	11	DSP	(1,1,0,1)	(4,1,4,2)	—
	1Regs	1	1724317	10	DSP	(1,1,0,1)	(5,0,4,2)	—
		2	2057045	10	DSP	(1,1,0,1)	(5,0,4,2)	—
Matrix	2Regs	1	1661455	10	DSP	(1,1,0,1)	(5,1,4,3)	(i)+(ii)+(iii)+(iv)
		2	1924000	11	DSP	(1,1,0,1)	(5,1,4,3)	(i)+(ii)+(iii)+(iv)
	1Regs	1	1978291	11	DSP	(1,1,0,1)	(6,0,4,3)	(i)+(ii)+(iii)+(iv)
		2	2189445	11	DSP	(1,1,0,1)	(6,0,4,3)	(i)+(ii)+(iii)+(iv)

T [ns]: Clock period of the synthesized processor core
#FUs: (#ALUs, #Shifters, #Multipliers, #MACs)
#Regs: (#General_registers(INT),#General_registers(LONG), #Data_pointers, #Loop_registers)
Addressing unit (i): no operation, (ii): post increment, (iii): post decrement, and (iv): index add

参考文献

[1] H. Akaboshi and H. Yasuura, "COACH: A computer aided design tool for computer architects," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E76-A, no. 10, pp. 1760-1769, 1993.

[2] N. N. Binh, M. Imai, A, Shiomi, and N. Hikichi, "A hardware/software partitioning algorithm for designing pipelined ASIPs with least gate count," in *Proc. 33rd DAC*, pp. 527–532, 1996.

[3] D. Gajski, N. Dutt, A. Wu, and S. Lin, *High-Level Synthesis*, Kluwer Academic Publishers, 1992.

[4] 濱辺雅哉, 能勢敦, 戸川望, 柳澤政生, 大附辰夫, "パイプラインプロセッサのハードウェア記述自動生成手法," 信学技報, VLD97-117, 1998.

[5] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan-Kaufman, 1990.

[6] I.-J. Huang and A. M. Despain, "Synthesis of instruction sets for pipelined microprocessors," in *Proc. 31st DAC*, pp. 5–11, 1994.

[7] 井上昭彦, 富山宏之, 石原亨, 安浦寛人, "組み込みシステムの低消費エネルギー化を目的としたデータ語長最適化手法," 信学技報, VLD98-29, 1998.

[8] 川崎隆志, 戸川望, 柳澤政生, 大附辰夫, "ディジタル信号処理向けプロセッサの自動調合成システムにおける並列化コンパイラ," 信学技報, VLD97-116, 1998.

[9] P. Lapsley, J. Bier, A. Shoham, and E. A. Lee, *DSP Processor Fundamentals: Architectures and Features*, Berkeley Design Technology, Inc., 1994–1996.

[10] V. K. Madisetti, *Digital Signal Processors*, IEEE Press, 1995.

[11] NEC, 信号処理 LSI（DSP/音声）データブック, 1996.

[12] J. Sato, A. Y. Alomary, Y. Honma, T. Nakata, A. Shiomi, N. Hikichi, and M. Imai, "PEAS-I: A hardware/software codesign system for ASIP development," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E77-A, no. 3, pp. 483–491, 1994.

[13] N. Seshan "High VelociTI processing," *IEEE Signal Processing Magazine*, 1998.

[14] Texas Instruments, *TMS320C54X DSP CPU and Peripherarl Reference Set*, Volume I, 1998.

[15] 戸川望, 桜井崇志, 柳澤政生, 大附辰夫, "ディジタル信号処理向けプロセッサのハードウェア/ソフトウェア協調合成システム," 信学技報, VLD97-115, 1998.

盛光印刷所